On the Dependence of Performance of Multiprocessors on Problem Size and Number of Processors

King Lee[1]

Report RNR-90-010, July 1990

NAS Systems Division
NASA Ames Research Center, Mail Stop T-045-1
Moffett Field, CA 94035

July 10, 1990

[1]The mailing address for Lee is Computer Science Dept., California State University, Bakersfield, CA 93309. This work is supported through NASA Contract NAS 2-12961

On the Dependence of Performance of Multiprocessors on Problem Size and Number of Processors

King Lee [1]

NAS Systems Division
NASA Ames Research Center
Moffet Field, CA 94035

July 10, 1990

Abstract

The 'Amdahl's law ' effect on parallel processors seriously limits the performance of computers when the number of processors increases on problems of fixed size. These effects can be avoided if the problem size increases sufficiently. A model is presented which can be used to predict quantitatively how the problem size must increase in order to maintain a given level efficiency as the number of processors increases. This model may also be useful in comparing the performance of radically different architectures.

Keywords: Parallel processing, performance modeling for parallel processors, parallel processor efficiency, relative performance of parallel processors.

[1]The mailing address for Lee is Computer Science Dept., California State University, Bakersfield, CA 93309. This work is supported through NASA Contract NAS 2-12961, while the author was an employee of Computer Sciences Corporation

1 Introduction

As computer technology matures it seems evident that we are approaching the limits of at which semiconductor devices can switch. In order to solve problems that are still intractable with present day supercomputers computer architects are turning to massive parallelism. However there is doubt as to whether massive parallelism is a feasible solution.

In [1] Amdahl computed the speedup R of multiprocessor computers as

$$R_1 = \left(F + \frac{1-F}{p}\right)^{-1}$$

where F is the fraction of the workload that is computed sequentially and p is the number of processors. The maximum speedup (ignoring communication and synchronization effects) occurs when $F = 0$ where the slope is $-p(p-1)$. This implies that it is very difficult to get close to maximal speedup with a large number processors when there is any sequential workload. The above expression and its consequences are known as Amdahl's law. When communication and synchronization costs are considered, under certain general conditions, there may be a limit to the number of processors that a problem can profitably utilize (see [2], [3] [10]). Using more processors would actually slow down the computation. This "maximum speedup" effect takes place even in the absence of sequential operations. Hack in [6] investigated in detail the "maximal speedup" effect and the marginal effects on performance of increasing processors.

Gustafson, Montry, and Benner in [5] reported speedups of the order of 1000 for a multiprocessor with 1024 nodes in solving scientific problems. These results seemed to be inconsistent with Amdahl's law and the "maximum speedup" effect. It was pointed out that when problems were run with a large number of processors the problem size usually increased, often in such a way as to keep the total running time constant. Gustafson, Montry, and Benner proposed a model where the parallel workload was proportional to the number of processors. Under this model they derived the following expression for the scaled speedup:

$$R_2 = F + (1-F)p$$

which has a more moderate slope $-p$ at $F = 0$.

Van-Catledge [9] considered several general models of computations in which the speedup was computed as a function of problem size. He considered a workload M with a sequential fraction F' and a parallelizable fraction $1-F'$. When the problem size increases, his simplest model assumes that the sequential workload remains $F'M$ and the parallelizable workload becomes $k(1-F')M$, where k is a measure of the increase in problem size. He derived the following expression for the scaled speedup

$$R_3 = \frac{F' + k(1-F')}{F' + k(1-F')/p}$$

This expression reduces to Amdahl's expression when $k = 1$ and to Gustafson, Montry and Benner's when $k = p$. He also found that for large values of k (≈ 4096) that indeed one can get very high speedup even with large F'. In this way he unites Amdahl's and Gustafson's expression for speedup and resolves the apparent inconsistency of Gustafson's results with Amdahl's law. Van-Catledge goes on to compare classes of computers using average hardware parameters.

Flatt and Kennedy in [3] investigated the effects of synchronization and communication on the performance and efficiency of parallel processors. They derive bounds on the performance under various synchronization cost functions, and they considered the effect of increasing the number of parallelizable operations directly with the number of processors. They found that as problem size increases that one can get close to linear speedup, but that the time is likely to increase. Worley in [10] obtains similar results using by using information theoretic methods. He found that for a large class of partial differential equations, increasing problem size will increase the computation time, and that the same arguments can be extended to other scientific problems as well. Patton in [8] gives a survey of work by several other authors who have considered this problem.

It seems that the effects of sequential operations and communication and synchronization on speedup are real but in some cases avoidable by increasing problem size. This paper focuses on how performance is related to problem size and number of processors. Many of our results overlap those of the authors cited above. The aim of this paper is to develop a quantitative model that will allow us to determine how much the problem size must be increased when we increase the number of processors in order to avoid the effects of

Amdahl's law. We shall also examine how changing the hardware and software parameters may influence the scaling requirements. A secondary goal is to develop a framework for the comparing the performance of computers with radically different architectures in a meaningful way.

In the next section we shall analyze a simple model and derive analytic expressions for the performance, efficiency and cost as a function of the number of processors and problem size. It turns out the general case will have the same qualitative behavior as the simple case. In the the third section we use constant performance and constant efficiency curves to gain insight on how problem size must be related to number of processors in order to avoid the effects of Amdahl's law. The fourth section applies the model to more realistic cases.

We shall follow the model by Van-Catledge in [9] using the techniques developed by Hockney and Jesshope in [7]. We assume that we have a computation with S sequential operations and K parallelizable operations. We shall assume that when the problem size varies, only K varies. This is a reasonable assumption in cases when increasing the size of the computation means increasing the limits of loops. We gather below for easy reference a list of expressions that will be used; the precise meaning of each of the expressions will be defined in the following sections.

p	$=$	number of processors used in the computation
K	$=$	number of operations in the parallizable part of the algorithm.
S	$=$	the sequential cost expressed in units of equivalent parallel operations
$C(p, K)$	$=$	the communication or synchronization cost expressed in units of equivalent parallel operations.
$k_{1/2}$	$=$	a measure of the how quickly one can achieve the peak performance.
	$=$	$(S + C(p, K))p$
τ	$=$	average time to perform one parallel operation by a processors.
τ_S	$=$	average time to perform one sequential operation.
τ_C	$=$	average time to perform one communication and sychronization operation.

$$\begin{aligned}
r_\infty^{(K)} &= \text{maximum achievable performance (in MFLOPS) for problem of size K} \\
r_\infty^{(p)} &= \text{the maximum achievable performance (in MFLOPS) with p processors} \\
&= p/\tau \\
r(p,\tau,C,S,K) &= \text{performance of computer with p processors, average time per operation } \tau\text{, C communication cost, S serial operations, K parallel operations} \\
T(p,\tau,C,S,K) &= \text{time to complete computation with p processors, average time per operation } \tau\text{, C communication cost, S serial operations, K parallel operations} \\
R &= \text{speedup} \\
&= \frac{T(1,\tau,0,S,K)}{T(p,\tau,C,S,K)} \\
E &= \text{Efficiency} \\
&= \frac{T(1,\tau,0,S,K)}{pT(p,\tau,C,S,K)} \\
&= \frac{\tau}{p} r(p,\tau,C,S,K)
\end{aligned}$$

2 Simple Model with Constant Communication Cost

The time to perform one parallel operation may be considerably different than time to perform one communication or even one sequential operation. We interpret τ to be the average time to perform a parallel operation. Let S' be the actual number of sequential operations each taking τ_S time, and C' be the actual number of operations for communications and synchronization each taking an average of τ_C time. Then the total overhead time $t_{overhead}$ is

$$\begin{aligned}
t_{overhead} &= \tau_S S' + \tau_C C' \\
&= \tau(\frac{\tau_S}{\tau} S' + \frac{\tau_C}{\tau} C') \\
&= \tau(S + C).
\end{aligned}$$

$S + C$ depend both on the number of sequential and communication operations and the ratios τ_S/τ and τ_C/τ. If we say that one actual sequential operation is equivalent to two parallel operations if $t_S = 2\tau$, then $S + C$ represent the number of sequential and communication operations in terms of equivalent parallel operations.

In this section we shall assume that the time required for sending data between processors, measured in equivalent parallel operations, is constant ($C(p, K) = C$). While this assumption is unrealistic, it will allow us to derive analytic results which gives insight into the qualitative behavior of multiprocessor systems. We shall investigate more realistic cases later.

The time to perform a computation with p processors, average time to complete an arithmetic operation τ, S sequential operations, C communication operations, and K parallelizable operations is given by

$$\begin{aligned} T(p, \tau, C, S, K) &= \tau(S + C + \frac{K}{p}) \\ &= \frac{(k_{1/2} + K)}{r_\infty^{(p)}}, \end{aligned} \tag{1}$$

where $k_{1/2} = (S+C)p$, and $r_\infty^{(p)} = p/\tau$. We shall see that $k_{1/2}$ is analoguous to the synchronization parameter $s_{1/2}$ discussed in [7]. Its interpretation is that the time required to compute the $k_{1/2}$ operations in parallel with p processors is the same as the time to compute $S + C$ operations sequentially.

Gustafson, Montry and Benner in [5] pointed out that when we increase the number of processors we usually also increases the problem size, often in such a way that total time remains constant. If we fix time to be T_0 in equation 1 and solve for K, we get

$$K = (T_0/\tau - (S + C))p \tag{2}$$

and we see that for this model it is possible to keep time constant while scaling linearly. Table 1 contains calculated values of $T(p, \tau, C, S, K)$ for various values of (p, K) when $S = \tau = 1$ and $C = 0$.

Performance is the number of operations divided by the time to perform the computation, so

$$\begin{aligned} r(p, \tau, C, S, K) &= \frac{(S + K)}{T(p, \tau, S, C, K)} \\ &= \frac{p(S + K)}{\tau((C + S)p + K)} \end{aligned} \tag{3}$$

K	p									
	4	8	16	32	64	128	256	512	1024	2048
2048	513.0	257.0	129.0	65.0	33.0	17.0	9.0	5.0	3.0	2.0
1024	257.0	129.0	65.0	33.0	17.0	9.0	5.0	3.0	2.0	1.5
512	129.0	65.0	33.0	17.0	9.0	5.0	3.0	2.0	1.5	1.3
256	65.0	33.0	17.0	9.0	5.0	3.0	2.0	1.5	1.3	1.1
128	33.0	17.0	9.0	5.0	3.0	2.0	1.5	1.3	1.1	1.1
64	17.0	9.0	5.0	3.0	2.0	1.5	1.3	1.1	1.1	1.0
32	9.0	5.0	3.0	2.0	1.5	1.3	1.1	1.1	1.0	1.0
16	5.0	3.0	2.0	1.5	1.3	1.1	1.1	1.0	1.0	1.0
8	3.0	2.0	1.5	1.3	1.1	1.1	1.0	1.0	1.0	1.0
4	2.0	1.5	1.3	1.1	1.1	1.0	1.0	1.0	1.0	1.0

Table 1:

Time as function of K and p

$$\tau(S + C + \tfrac{K}{p})$$

$$C = 0 \qquad S = 1 \qquad \tau = 1$$

$$= r_\infty^{(p)} \frac{S+K}{k_{1/2}+K}.$$

Table 2 contains calculated values of $r(p,\tau,C,S,K)$ for various values of (p,K) when $S=\tau=1$ and $C=0$.

If we fix a value of p and let $K \to \infty$, $r(p,\tau,C,S,K)$ will approach the asymptotic performance $r_\infty^{(p)}$. Let f be the fraction of the asymptotic performance $r_\infty^{(p)}$ obtained at (p,K). If we solve $fr_\infty^{(p)} = r(p,\tau,C,S,K)$ for K using equation (3) we get

$$K = \frac{f(S+C)p - S}{1-f}$$

and a performance of one half $r_\infty^{(p)}$is achieved when $f = 1/2$ or

$$\begin{aligned} K &= (S+C)p - 2S \qquad (4) \\ &= k_{1/2} - 2S. \end{aligned}$$

K	p									
	4	8	16	32	64	128	256	512	1024	2048
2048	4.0	8.0	15.9	31.5	62.1	120.5	227.7	409.8	683.0	1024.0
1024	4.0	7.9	15.8	31.1	60.3	113.9	205.0	341.7	512.5	683.3
512	4.0	7.9	15.5	30.2	57.0	102.6	171.0	256.5	342.0	410.4
256	4.0	7.8	15.1	28.6	51.4	85.7	128.5	171.3	205.6	228.4
128	3.9	7.6	14.3	25.8	43.0	64.5	86.0	103.2	114.7	121.4
64	3.8	7.2	13.0	21.7	32.5	43.3	52.0	57.8	61.2	63.0
32	3.7	6.6	11.0	16.5	22.0	26.4	29.3	31.1	32.0	32.5
16	3.4	5.7	8.5	11.3	13.6	15.1	16.0	16.5	16.7	16.9
8	3.0	4.5	6.0	7.2	8.0	8.5	8.7	8.9	8.9	9.0
4	2.5	3.3	4.0	4.4	4.7	4.8	4.9	5.0	5.0	5.0

Table 2:

Performance as a function of p and K

$r(p, \tau, C, S, K)$

$C = 0 \qquad S = 1 \qquad \tau = 1$

This value of $k_{1/2}$ determines how rapidly one reaches the asymptotic performance. In Table 2, $C = 0$ and $S = 1$ and for each p, one achieves half the asymptotic performance for that p along the line $K = p - 2$.

If we hold K constant in equation (3) and let $p \to \infty$, we see that the performance tends to

$$r_{\infty}^{(K)} = \frac{S + K}{\tau(S + C)} \tag{5}$$

The reason that $r_{\infty}^{(K)}$ does not increase without bound as p increases is that even if both the time spent computing the parallel operations go to zero, the time spent computing the sequential and communication part remains constant. If $S + C \neq 0$ problems cannot be scaled up indefinitely by just increasing p. Let f' be the fraction of the asymptotic performance $r_{\infty}^{(K)}$ obtained at (p,K). If we solve $f' r_{\infty}^{(K)} = r(p, \tau, C, S, K)$ for p using equation

K	p									
	4	8	16	32	64	128	256	512	1024	2048
2048	99.9	99.7	99.3	98.5	97.0	94.2	88.9	80.0	66.7	50.0
1024	99.7	99.3	98.6	97.1	94.2	89.0	80.1	66.7	50.0	33.4
512	99.4	98.7	97.2	94.3	89.1	80.2	66.8	50.1	33.4	20.0
256	98.8	97.3	94.5	89.2	80.3	66.9	50.2	33.5	20.1	11.2
128	97.7	94.9	89.6	80.6	67.2	50.4	33.6	20.2	11.2	5.9
64	95.6	90.3	81.3	67.7	50.8	33.9	20.3	11.3	6.0	3.1
32	91.7	82.5	68.8	51.6	34.4	20.6	11.5	6.1	3.1	1.6
16	85.0	70.8	53.1	35.4	21.2	11.8	6.3	3.2	1.6	0.8
8	75.0	56.3	37.5	22.5	12.5	6.6	3.4	1.7	0.9	0.4
4	62.5	41.7	25.0	13.9	7.4	3.8	1.9	1.0	0.5	0.2

Table 3:

Efficiency as a function of p and K

$$100\frac{r(p,\tau,C,S,K)}{p}$$

$$C=0 \quad S=1 \quad \tau=1$$

(5) and (3) we get

$$p \;=\; \frac{f'K}{(S+C)(1-f').}$$

We achieve half the asymptotic performance $r_\infty^{(K)}$ when $f' = 1/2$ where we have $p = K/(S+C)$ and $(S+C)p = k_{1/2} = K$.

The performance increases most rapidly in the direction of the gradient

$$\left(\frac{\partial r}{\partial p}, \frac{\partial r}{\partial K}\right) \;=\; \left(\frac{K(S+K)}{\tau((S+C)p+K)^2}, \frac{Sp(p-1)+Cp^2}{\tau((S+C)p+K)^2}\right) \qquad (6)$$

The gradient has slope 1 along one arm of the hyperbola

$$\left(K+\frac{S}{2}\right)^2 - (S+C)\left(p-\frac{S}{2(S+C)}\right)^2 \;=\; \frac{S^2}{4}(1-\frac{1}{(S+C)})$$

K	p									
	4	8	16	32	64	128	256	512	1024	2048
2048	1.0	1.0	1.0	1.0	1.0	1.1	1.1	1.2	1.5	2.0
1024	1.0	1.0	1.0	1.0	1.1	1.1	1.2	1.5	2.0	3.0
512	1.0	1.0	1.0	1.1	1.1	1.2	1.5	2.0	3.0	5.0
256	1.0	1.0	1.1	1.1	1.2	1.5	2.0	3.0	5.0	9.0
128	1.0	1.1	1.1	1.2	1.5	2.0	3.0	5.0	8.9	16.9
64	1.0	1.1	1.2	1.5	2.0	3.0	4.9	8.9	16.7	32.5
32	1.1	1.2	1.5	1.9	2.9	4.8	8.7	16.5	32.0	63.0
16	1.2	1.4	1.9	2.8	4.7	8.5	16.0	31.1	61.2	121.4
8	1.3	1.8	2.7	4.4	8.0	15.1	29.3	57.8	114.7	228.4
4	1.6	2.4	4.0	7.2	13.6	26.4	52.0	103.2	205.6	410.4

Table 4:

Cost as a function of p and K

$$\frac{1}{\text{Efficiency}}$$

which is asymptotic to the line $K = \sqrt{S + Cp}$. In the region in the (p, K) plane above this hyperbola the gradient has slope < 1 and we increase performance faster by increasing p than by increasing K. We shall refer to this region as the region where parallel operations dominate. In Table 2 it corresponds to the points above $K = p$. In the region below the hyperbola the gradient has slope > 1 and we increase performance faster by increasing K than by increasing p. We shall refer to this region as the region where sequential operations dominate. In Table 2 it corresponds to the points below $K = p$.

The speedup R is

$$\begin{aligned} R &= \frac{T(1, \tau, 0, S, K)}{T(p, \tau, C, S, K)} \\ &= \tau r(p, \tau, C, S, K) \end{aligned}$$

and is just a multiple of the performance. We prefer to use performance rather than speedup because performance allows more meaningful comparisons between different computers such as the Cray YMP with 8 processors and the Connection Machine with 64K processors.

For $p > 1$ the efficiency is given by

$$\begin{aligned} E &= \frac{T(1,\tau,0,S,K)}{pT(p,\tau,C,S,K)} \\ &= \frac{S+K}{(S+C)p+K} \end{aligned} \tag{7}$$

E tends to 1 as K tends to infinity and E tends to 0 as p tends to infinity. If we solve for K in terms of E and p in equation (7) we get

$$K = \frac{E(S+C)p - S}{1-E.} \tag{8}$$

The set of points where $E = const$ in the (p,K) plane are lines with slope equal to $(S+C)E/(1-E)$. Table 3 computes the efficiency for $S = \tau = 1$ and $C = 0$. The gradient of the efficiency is

$$\left(\frac{\partial E}{\partial p}, \frac{\partial E}{\partial K}\right) = \left(\frac{-(S+C)(S+K)}{((S+C)p+K)^2}, \frac{(S+C)p-S}{((S+C)p+K)^2}\right). \tag{9}$$

It is evident from the quadratic term in the denominator of $\partial E/\partial p$ that efficiency drops off rapidly as p increases for large p.

Let B be the cost of a single processor (in dollars). Then the cost of the computation in terms of dollars per MFLOPS for is given by

$$\begin{aligned} \text{Cost} &= \frac{Bp}{r(p,\tau,C,S,K)} \\ &= \frac{B\tau}{E} \end{aligned}$$

Thus the cost of performing the given computation is inversely proportional to the efficiency (Table 4). The absolute value of the gradient of cost is inversely proportional to the square of E, and increases very rapidly when E is small. Another commonly used measurement of cost is MFLOPS per dollar. This is simply the inverse of B, and is directly proportional to E.

3 Discussion of the Simple Model

In visualizing the MFLOPS surfaces defined by equation (3) it will be useful to consider curves $r(p, \tau, C, S, K) = const$ in the (p, K) plane. Figure 1 shows the curves of constant performance for $r = 8$, $r = 32$, and $r = 128$ when $\tau = 1$, $S = 1$ and $C = 0$. Also shown is the line $K = \sqrt{S + Cp}$ (which asymptotically divides the region where sequential operations dominates from the region where parallel operations dominate), and the lines of constant efficiency for $E = .50$, $E = .90$, and $E = .95$. Note that we are using a log log scale for the axes. The constant performance curves form a family of rotated hyperbolas whose vertices are asymptotic to the line $K = \sqrt{S + Cp}$. Figure 2 shows the is a similar curve when $\tau = 1$, $S = 1$ and $C = 2$. Increasing the overhead $S + C$ corresponds either to increasing the number of overhead operations involved or to increasing the ratios τ_S/τ and/or τ_C/τ. The effect is to raise the curves, i.e. to increase $k_{1/2}$.

Adding processors, which is equivalent to moving to the right on a horizontal line in the (p, K) plane, will always speed up performance in this simple model. Depending on the region in the (p, K) plane the improvement in performance may be slow or rapid. When we are in the region where parallel operations dominate increasing p

- increases performance significantly (Table 2),
- decreases time significantly (Table 1),
- raises costs (in units of dollars per MFLOPS) moderately (Table 4).

In the region where sequential operations dominate increasing p

- increases performance slowly,
- decreases time slowly,
- raises costs rapidly.

We can find the value of p, $p_{1/2}$, at which the the efficiency is .5 by setting E to .5 in equation (7) and solving for p. We find K

$$p_{1/2}^{(K)} = \frac{K + 2S}{S + C}.$$

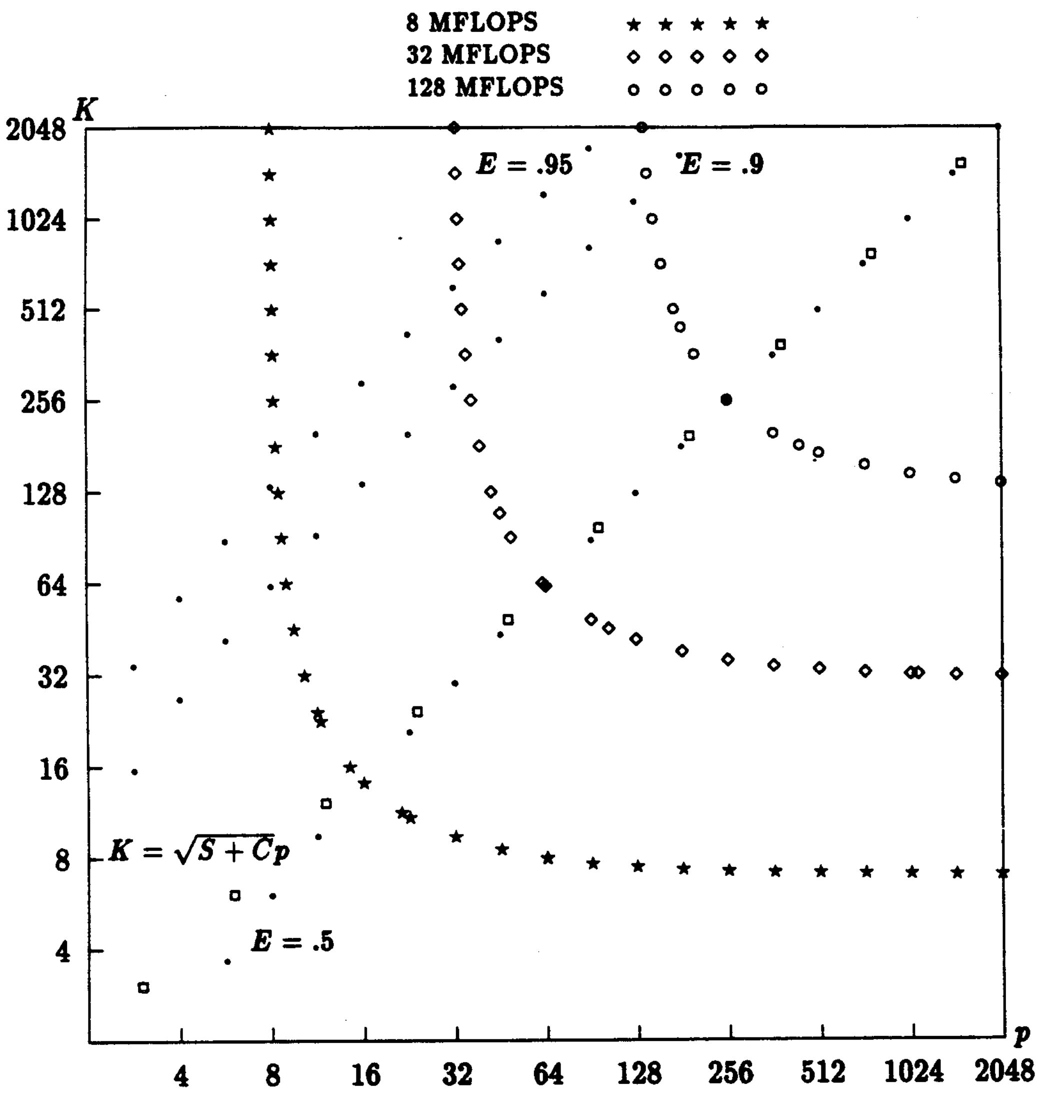

Figure 1:

Constant Performance and Constant Efficiency Curves

$C = 0 \quad S = 1 \quad \tau = 1$

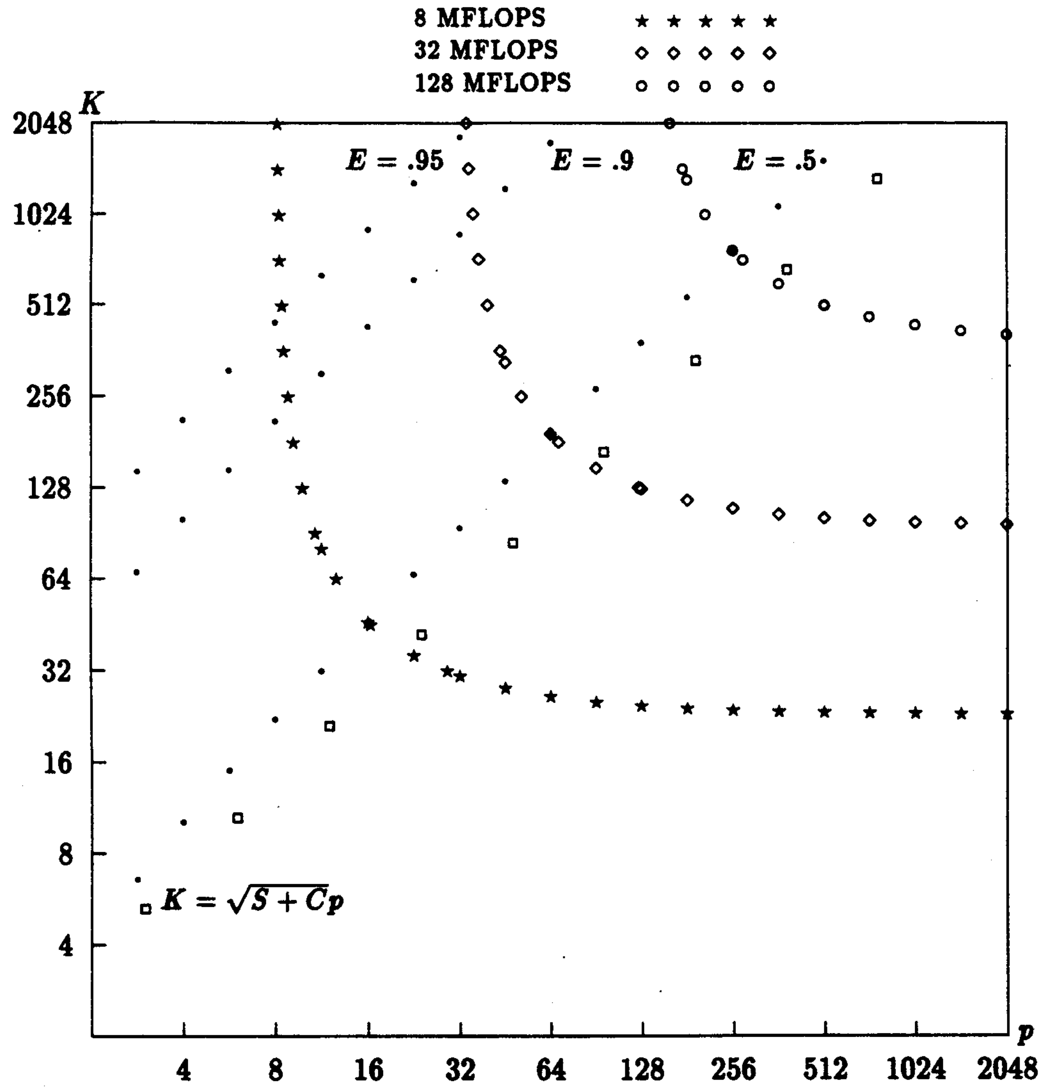

Figure 2:

Constant Performance and Constant Efficiency Curves

$C = 2 \quad S = 1 \quad \tau = 1$

After solving for p in equation (8), we have

$$\begin{aligned}\frac{p^K_{1/2}}{p} &= \frac{K+2S}{S+C}\frac{(S+C)E}{(1-E)K+S} \\ &= \frac{E(1+\frac{2S}{K})}{1-E+\frac{S}{K}} \\ &\approx \frac{E}{1-E}\end{aligned}$$

for large K. It follows that if we are computing with efficiency E at a point (p,K) for large K and we increase the number of processors by a factor much greater $E/(1-E)$ we shall operate at less than .5 efficiency. This can also be deduced from from Figure 1 where the lines of constant efficiency are parallel to line $E = .5$. Thus if we have an efficiency of .80, we may increase the number of processors by a factor of $E/(1-E) = 4$ before the efficiency drops to about .5. Further increases in p would increase performance slowly. On the other hand, if we have an efficiency of .95 using p processors, and we can increase the number of processors by a factor of $E/(1-E) = 19$ before the efficiency drops to .5. Thus efficiency gives information not only on the utilization of the processors, but also on the potential for speedup that can be obtained by adding processors.

From equation (2) the slope of the constant time curves takes on all values from 0 to ∞ as the time T_0 varies from $\tau(S+C)$ to ∞. Similarly from equation (8) the slope of the constant efficiency curves also take on all values from 0 to ∞ as E varies from 0 to ∞. Therefore if we vary K linearly with p with any slope, the corresponding curve will be asymptotic to a constant time curve and a constant efficiency curve on a log log scale. Suppose that we have a computation of size K_0 with p_0 processors. Let

$$\begin{aligned}T_0 &= \tau(S+C+\frac{K_0}{p_0}) \\ E_0 &= \frac{(S+K_0)}{(S+C)p_0+K_0}\end{aligned}$$

from equations (1) and (7). Suppose we wish to increase the problem size and number of processors in such a way as to keep the time at T_0. Then from equation (10)

$$T_0 = \tau(S + C + \frac{K}{p})$$

and solving for K we get

$$\begin{aligned} K &= (\frac{T_0}{\tau} - (S + C))p \\ &= \frac{K_0}{p_0}p. \end{aligned}$$

If we substitute this line into to equation (7) we will find that along this line the efficiency is

$$\begin{aligned} E &= \frac{S + \frac{K_0}{p_0}p}{(S + C)p + \frac{K_0}{p_0}p} \\ &= (\frac{S}{p} + \frac{K_0}{p_0})/(S + C + \frac{K_0}{p_0}) \\ &= (\frac{s}{p} - \frac{s}{p_0} + \frac{s}{p_0} + \frac{K_0}{p_0})/(S + C + \frac{K_0}{p_0}) \\ &= E_0 - \frac{Sp_0}{(S + C)p_0 + K_0}\left(\frac{1}{p_0} - \frac{1}{p}\right) \end{aligned}$$

Therefore scaling in such a way as to keep time constant decreases efficiency; if S is small compared to C or K_0/p_0 then the decrease in efficiency will be small.

If we wish to increase p and K in such a way as to maintain the same efficiency, we see from equation (8) that we must stay on the constant efficiency line with slope $(S + C)E/(1 - E)$. If we increase the number of processors by Δp we must increase K by ΔK, where, from equation (8),

$$\Delta K = \frac{E}{1 - E}(S + C)\Delta p$$

If ΔK is less than this value efficiency drops. For example if $E = .99$, then in order to maintain the same efficiency we must have $\Delta K = 99(S + C)\Delta p$. On the other hand, if $E = .8$, then to maintain .8 efficiency we must have

$\Delta K = 4(S + C)\Delta p$. If we substitute equation (8) into equation (1) we find that the time on the constant efficiency curve to be

$$T = \tau(S + C + \frac{E(S+C)}{1-E} - \frac{S}{(1-E)p})$$

so that time increases with increasing p.

What is important is not maintaining constant efficiency, but to avoiding the region of low efficiency. The user may decide the minimal acceptable efficiency; once that is determined the user is constrained to a region in the (p, K) plane that is (asymptotically) bounded by a line. We shall see in the next section that for more complex models this region may be bounded by a curve.

Up to now we have let K, the number of parallelizable operations, represent problem size. This may not be the best representation of problem size for two reasons. The first reason is that the end user may find another measure of problem size more natural than number of operations. If the problem at hand involves square matrices, the user may find N, the dimension of the matrices, or N^2, the size of the matrices, to be more convenient to work with. If we can express the number of parallelizable operations as a function of N, then we can let the vertical axis represent N or N^2.

The second reason is that the number of operations may not be appropriate when comparing performance on computers with radically different architectures. When comparing different architectures , we sometimes find that the algorithm which is suitable for one architecture may not be suitable for another architecture. On some computers users even trade off communication time for computation time by recomputing data rather than saving it. In this case it seems appropriate to replace K by a parameter related to the intrinsic problem size and not to the number of operations. For problems involving square matrices, this parameter might be the dimensions of the matrix N.

When comparing different parallel computers one should not consider only peak performance, but performance as a function of both p and K. A natural way of comparing performances is is to compare the constant performance curves and and constant efficiency lines of the two computers. Where the computers differ greatly in the number of processors, we can scale the p axis to make the curves more comparable. For example, if one were to compare the CM-2 Connection Machine with 64K processors with the iPSC/i860 with

128 processors, one might plot the p axis for the Connection machine in units of Lp processors. One might choose L to be 32 because 32 processors share one floating point chip, or one might choose L to be ratio of the cost (in dollars) of a processor of the iPSC/i860 and the cost of a processor of the CM-2.

In Figure 3 we show the constant performance curve for a computer that is twice as slowly ($\tau = 2$) as the computer in Figure 1. We assume that the $(S + C)$ is the same for both figures so, if the number of actual overhead operations remains the same, the time τ_S and t_C also doubles. Let us assume this machine of Figure 3 costs half as much per processor and we can afford twice as many processors. In Figure 4 we plot the (extended) data with the p axis scaled by one half. Intuitively we expect that the computer with the slower processors can achieve the same asymptotic performance $r_\infty^{(p)}$ as the computer with faster processors by using twice as many processors. Intuition is correct as we can see by considering considering the vertical line $p = 8$ in Figure 1 $p = 16$ in Figure 4. Note however that it takes a larger problem to achieve half of asymptotic performance $r_\infty^{(p)}$. This is expected from equation (4) where we saw that the value of K which gave one half the asymptotic performance $r_\infty^{(p)}$ was proportional to p.

For a problem of a given size the faster machine has twice the asymptotic performance $r_\infty^{(K)}$ of the slower machine. This is evident by comparing the horizontal asymptotes in Figures 1 and 4. It also follows from factor τ in the denominator of $r_\infty^{(K)}$ in equation (5). The reason is that with a large number processors the time is dominated by the sequential operations, and on the sequential operations the computer with the faster processor will do better.

If we compare Figures 4 and 1 we see that doubling p and τ (while keeping (S+C) constant) and rescaling the p axis is similar to increasing C. We may note that the lines of constant efficiency in Figure 4 lies above those in Figure 1. This also follows from equation (7). If we have a choice of cutting in half τ together with t_S and t_C or doubling the number of processors we should opt for cutting in half the average time to perform an operation, others things being equal. If we decrease τ without also decreasing τ_S and τ_C proportionately the effect would have been to increase $r_\infty^{(p)}$ and also $k_{1/2}$ and the result may be similar to using twice as many slow processors.

Recall that τ is the average time to complete one operation and not the cycle time of a computer. τ includes the effect of special characteristics of the computer system such as cache, compiler optimization, vectorization,

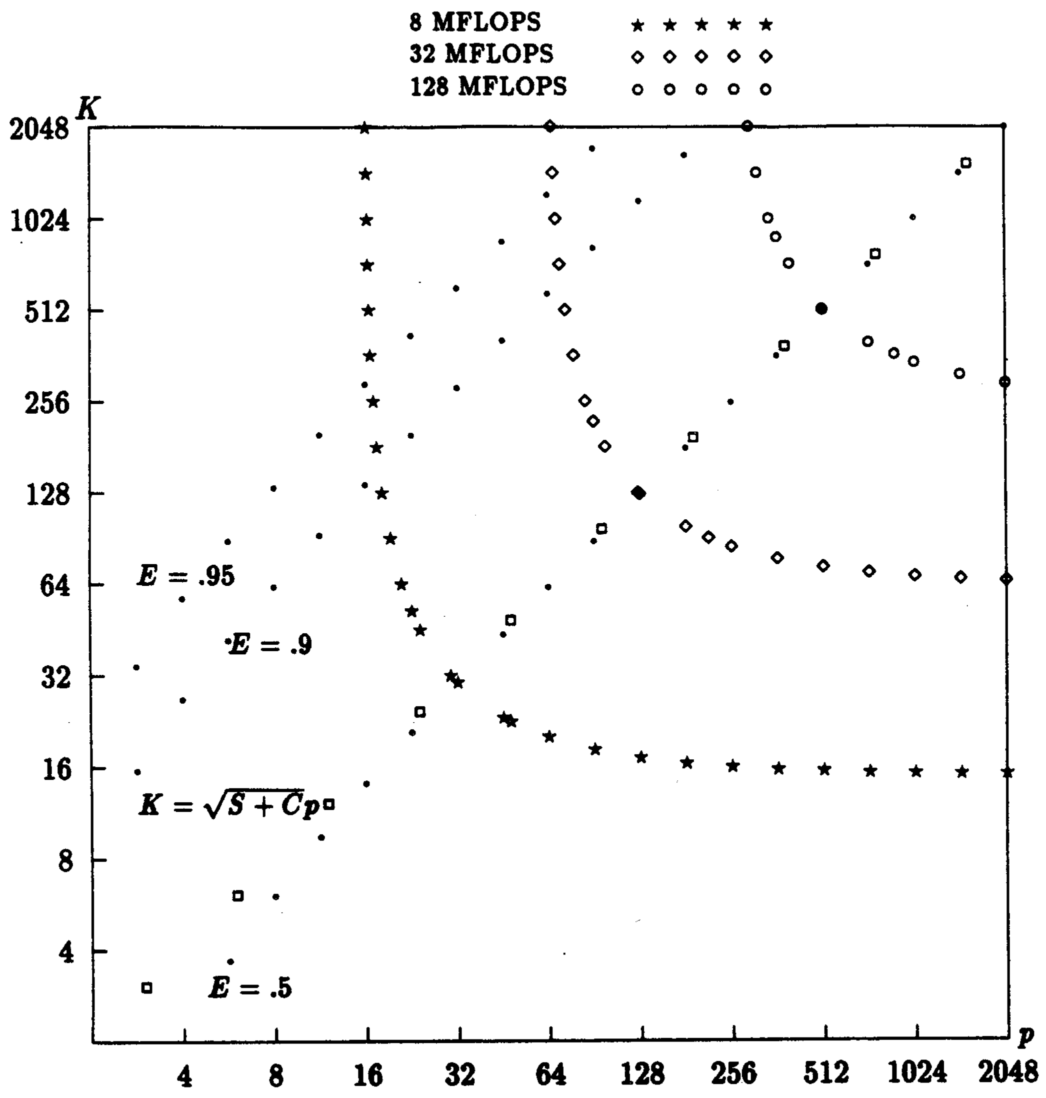

Figure 3:

Constant Performance and Constant Efficiency Curves

$C = 0 \quad S = 1 \quad \tau = 2$

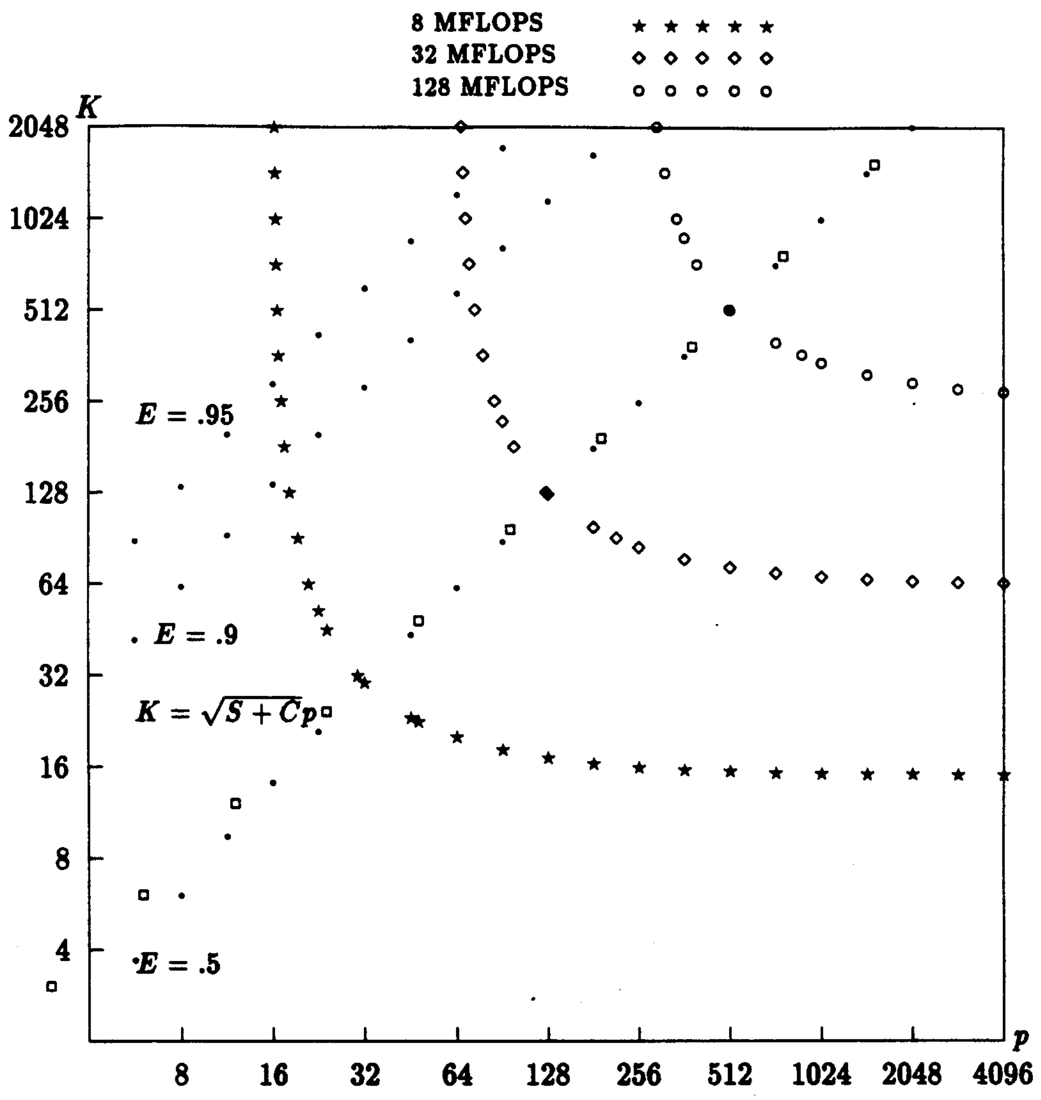

Figure 4:

p axis rescaled
Constant Performance and Constant
Efficiency Curves

$C = 0 \quad S = 1 \quad \tau = 2$

memory bandwidth, and communication costs. The information contained in the constant performance curves takes all these features into account and one may be able to infer values of τ from the vertical asymptotes of the figures based on experimental measurements. One would expect that computers with fast processors would outperform computers with a large number of slow processors for small problems. The constant performance curves allows one to determine the region where one computer outperforms the other. If one recalls that the lines of constant efficiency are also the lines of constant cost, one may also deduce in which regions the cost of the computation is less for one computer compared with another computer.

For any real computer we have finite memory and number of processors. This means we cannot run programs over a certain size K_{max} and if we have only p_{max} processors our performance is limited to $r_{\infty}^{p_{max}}$. We are constrained to compute in the region bounded by the p axes, K axes, the line $K = K_{max}$ and the line $p = p_{max}$. The greatest performance within this rectangle occurs at (p_{max}, K_{max}). If we wish to obtain a level M of performance, we must be sure that a portion of the curve $r(p, \tau, C, S, K) = M$ lies in that rectangle. Recall the line $K = \sqrt{S + Cp}$ asymptotically separates the region where sequential operations dominate from the region where parallel operations dominate. If this line goes through (or near) (p_{max}, K_{max}) we say that our computer is balanced for this algorithm. If this line intersects the line $K = K_{max}$ we shall say we are memory bound; if this line intersects $p = p_{max}$ we shall say we are processor bound. The effects of Amdahl's law become evident if we are memory bound; in that case using all p_{max} processors would mean computing in the region where sequential operations dominate and we cannot obtain easily additional speedup by adding processors. We make good use of massive parallelism if we are processors bound; in that case it is possible for all p_{max} processors to be used (for sufficiently large problems) while remaining in the region where parallel operations dominate. The limitation of performance depends not only technology through p_{max} and K_{max} but also on the algorithm through the value of $S + C$ that appears in the slope of the line $K = \sqrt{S + Cp}$. If one must make tradeoffs between p_{max} and K_{max}, the best tradeoff will be the one such that (p_{max}, K_{max}) lies near the line $K = \sqrt{S + Cp}$. If we have a number of applications with widely varying values of $S + C$, then no one tradeoff will be optimal for all applications.

Currently VLSI technology is advancing rapidly and one might expect that class of computers based on VLSI to increase performance most rapidly providing there are no other bottlenecks. VLSI advances in memory technology should benefit all class of computers equally. VLSI should allow p_{max} to increase more rapidly for massively parallel computers than for conventional supercomputers. However one should not underestimate the growth potential of conventional architectures in the near future. The dominant supercomputer today, the Crays, have traded high peak performance for short vector startup time (small $n_{1/2}$ in the terminology of [7]). As problems increase in size peak performance becomes more important and vector startup time becomes less important. It is feasible to increase peak performance for the Crays (at the cost of increasing $n_{1/2}$) by using multiple pipes in the arithmetic functional units. Examples of computers which obtain high peak performance with multiple pipes per CPU are the Fujitsu and NEC computers. Replicating pipes within a functional unit is a form of parallelism which is limited by the ability to move data efficiently from memory to the pipes. We shall see in the next section the parallelism in replicating processors in a message passing architecture may be limited by the ability to efficiently move data from one processor to another.

In summary, we have found that it is important to scale in such a way as to avoid computing in the region of low efficiency. In order to increase p and K in such way to keep time constant one would increase K linearly with p; efficiency will drop, the amount of drop primarily depending on the sequential component of the computation. In order to increase p and K in such a way to keep efficiency constant one would again increase K linearly with p; scaling up while keeping efficiency constant will increase the time. Increasing CPU and overhead speed moves the constant performance curves down and to the left; increasing overhead $(S + C)$ tends to raise the constant performance curves. Computers with a large number of slow relatively processors will generally may have relatively large value of $k_{1/2}$. Available resources determine p_{max} and K_{max}. If tradeoffs have to be made, the values should be chosen so that the computer is balanced.

4 Model with Communication as a Function of p

In general the communication and synchronization costs $C(p, K)$ will be a function of K and p and

$$T(p,\tau,C,S,K) = \tau(S + C(p,K) + \frac{K}{p}) \tag{10}$$

$$r(p,\tau,C,S,K) = \frac{p}{\tau}\frac{(S+K)}{(C(p,K)+S)p+K}. \tag{11}$$

If $C(p,K) \to \infty$ as $p \to \infty$ the performance approaches 0 asymptotically. Therefore for fixed K, if $r(p,\tau,C,S,K) \geq 0$ and $r(0,\tau,C,S,K) = 0$ there is a value of p, $p^{(K)}_{max}$, for which the performance is a maximum and we have the "maximum speedup" effect. In general $C(p,K)$ will also increase with K but unless $C(p,K)$ grows faster than linear with K, K will dominate the denominator of equation (11). It is hard to imagine a real application where this would be the case. It can be shown that if $C(p,K)$ is $O(K^\alpha)$ for some $\alpha > 1$, that there is a maximum achievable performance. Even if $C(p,K)$ is $O(K^\alpha/p^\beta)$ for some $\alpha > 1$ and $\beta > 0$ equation (7) shows that for fixed p, the efficiency will tend to zero as $K \to \infty$, making it very hard to achieve high performance and high efficiency on realistic problems. We shall assume that $C(p,K)$ grows at most linearly with K in the remainder of this paper.

Let us consider first one case in detail. We choose the algorithm for multiplying two N by N matrices on a message passing hypercube type computer as described in [4]. The time it takes to perform the computation was derived to be

$$\begin{aligned} T &= \tau(C(p,K) + \frac{K}{p}) \\ &= \tau(C_0\sqrt{p}(\sqrt{p}-2) + C_1\frac{N^2}{\sqrt{p}} + \frac{2N^3}{p}) \end{aligned}$$

where C_0 is a constant proportional to the time to start sending a message and C_1 is a constant proportional to the transmission time. We have assumed that S, the sequential operations, are negligible compared to K and $C(p,K)$, and that $K = 2N^3$. It follows from equation (3) that

$$\begin{aligned} r(p,\tau,C,0,N) &= \frac{2N^3}{\tau(C_0\sqrt{p}(\sqrt{p}-2) + C_1\frac{N^2}{\sqrt{p}} + \frac{2N^3}{p})} \\ &= \frac{p}{\tau}\frac{2N^3}{C_0p\sqrt{p}(\sqrt{p}-2) + C_1N^2\sqrt{p} + 2N^3} \end{aligned}$$

Figure 5 gives a plot the constant performance and constant efficiency curves. For fixed N the performance goes to 0 as $p \rightarrow \infty$. and therefore there is a is a "maximum speedup" effect. Imagine that one moves right on a horizontal line, say along the line $K = 32$. Performance increases as we cross constant performance curves with increasing values until we touch one constant performance curve tangentially at $p = p'$. In this case $K = 32$ touches the constant performance curve at about 128 MFLOPS. As we continue to move right we shall recross constant performance curves but now with decreasing values. For $p > p'$ the curve has positive slope of about .3. This means that if we scale a problem in such a way that its curve on the (p, K) plane is a line with slope less than .3 there will be a maximum achievable performance because there will be some constant performance curves that it will never intersect. If we scale the with slope greater than .3 then we shall cross every constant performance curve and we can achieve any performance we wish provided p and K are sufficiently large.

If we fix p and increase N the performance approaches $r_\infty^{(p)}$ as $N \rightarrow \infty$. The steepness of the vertical asymptotes of Figure 5 compared to those of Figure 1 is due to the scaling of the vertical axis. Recall that $K = 2N^3$ so that $N = 4$ corresponds to $K = 128$.

The constant efficiency curves for this figure have slopes $\alpha < 1$. Let us approximate α by .5 (actually a value of .67 would be more accurate, but we use .5 for purposes of illustration). These curves are asymptotic to curves of the form $N = Ap^{.5}$ for some A. This means that if we increase the number of processors by 4, we need to increase N by 2 in order to keep the same efficiency. Stated another way, if we increase problem size (as measured by N) linearly with p we increase efficiency. If we scale the vertical axis in terms of N^2, a measure of the memory requirements, constant efficiency curves have the form $N^2 = A'p$. This means that if increase problem size (measured in terms of memory requirements) linearly with p we maintain

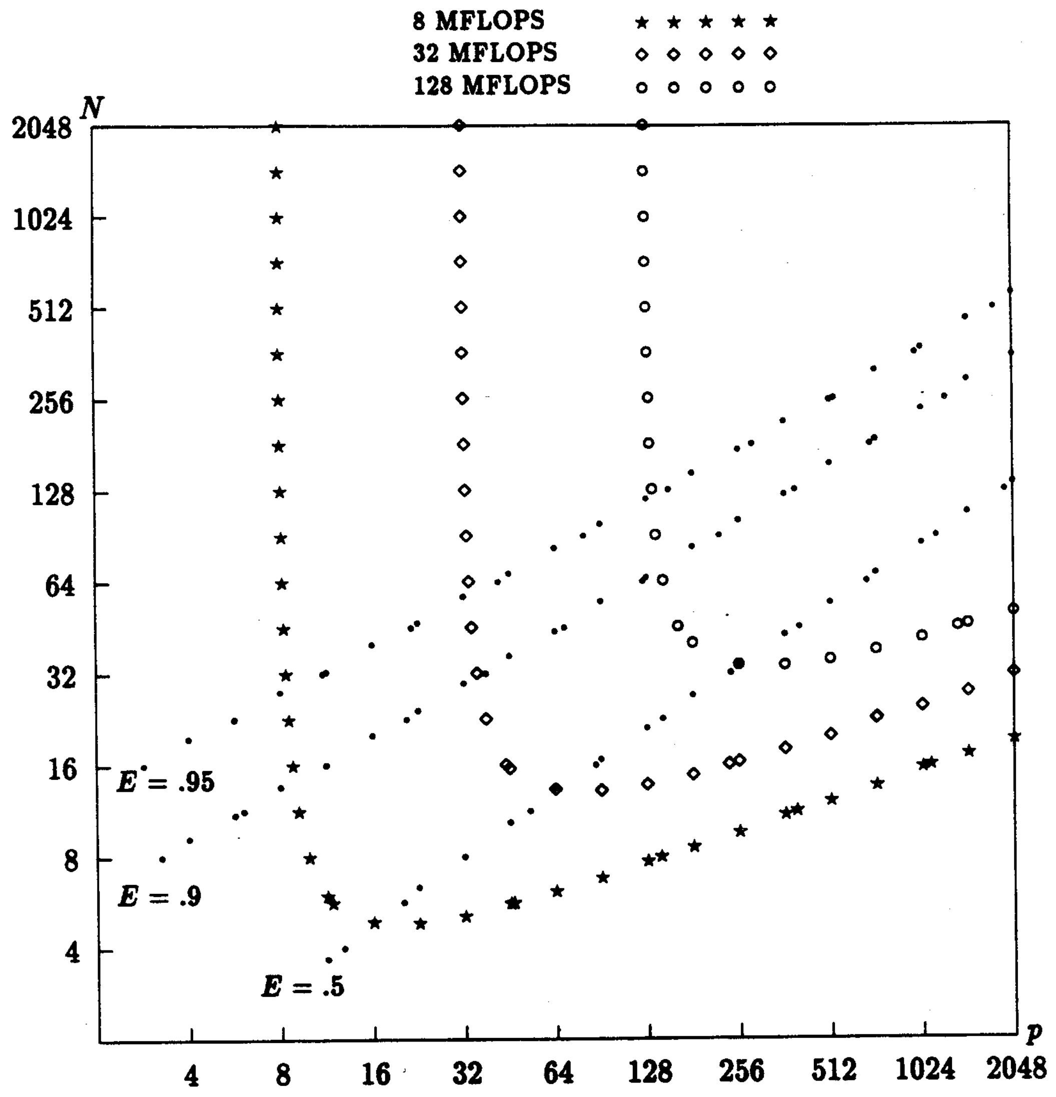

Figure 5:

Matrix Multiplication
Constant Performance and Constant
Efficiency Curves

$S = 0 \quad \tau = 1$

the same efficiency. Finally, if we scale the vertical axis in terms of K, the number of operations, we have $K = A''p^{1.5}$ so that if we increase problem size (in terms of number of operations) linearly with p, we decrease efficiency. The slope of the constant efficiency lines is of critical importance in determining how problems will scale. The slope depends in part on the choice of vertical scale, and there may be several equally valid choices.

Let us next consider a more general case. Different functions for $C(p, K)$ in equation (10) will lead to different constant efficiency curves, and therefore different scaling effects. Let us consider the case where $C(p, K) = O(p^\alpha)$ for some $\alpha > 0$. If we replace C by $C_p p^\alpha$ in equation (8) then the constant efficiency curves have the form

$$K \approx \frac{EC_p}{1-E} p^{1+\alpha}$$

for large p. It follows that if we increase K linearly with p we fall below that curve and efficiency will decrease. If we substitute the above expression of K into equation (10) using $C(p, K) = C_p p^\alpha$ the time along the constant efficiency curve is

$$T \approx \tau(S + Cp^\alpha + \frac{E(S+C)}{1-E} p^\alpha).$$

Therefore along constant efficiency curves time increases like p^α.

Suppose we wish to increase p and K in such a way that the time to perform the computation remains constant at T_0. Using equation (1) and again letting $C(p, K) = C_p p^\alpha$ for some $\alpha > 0$ we have for the constant time curve

$$T_0 = \tau(S + C_p p_0^\alpha + \frac{K_0}{p_0})$$

If we solve equation (10) for K and set T to T_0 we find that

$$K = (\frac{T_0}{\tau} - (S + C_p p^\alpha))p. \qquad (12)$$

K becomes 0 when p is so large that the time to perform the overhead sequential and communication operations equals the time to perform the original smaller computation. This means that if the overhead grows with the size of the problem, we cannot scale indefinitely in such a way as to keep the time constant. Along the constant time curve, the efficiency can be found by substituting equation (12) into equation (7) and we have

$$E = \frac{S + (\frac{T_0}{T} - (S + C_p p^\alpha))p}{(S + C_p p^\alpha)p + (\frac{T_0}{T} - (S + C_p p^\alpha))p}$$

which becomes 0 for sufficiently large p, as we would expect. Therefore if the overhead grows with increasing p, we must increase the time as we scale the problem up.

Suppose that $C(p, K)$ is $O(p^\alpha)$ for some $\alpha > 0$, and we wish to use some measure of problem size N instead of K. Supoose we have $K = O(N^\beta)$ for some $\beta \geq 1$. In the example of matrix multiplication discussed above, we had $\alpha = 1$ and $\beta = 3$. It can be shown that the constant efficiency curves of equation (8) becomes of the form

$$N \approx Ap^{\frac{1+\alpha}{\beta}}$$

for some A and large p. We shall say that if α is "large" the problem is communication intensive with respect to N; if β is "large" the problem is computation intensive with respect to N. If $(1 + \alpha)/\beta \leq 1$ then we shall say the computation scales well with respect to N. α and β are properties of the algorithm and and the algorithm therefore determines whether a problem will scale well. $C(p, K)$ is determined by the algorithm and in turn the choice of algorithm is influenced by the underlying physical architecture. If the problem scales well with respect to N, then we can obtain high efficiency we we increase N directly with p. The time, however, depends on K and may increase faster than linear if the problem is computationally intensive with respect N.

In summary, how fast $C(p, K)$ grows with increasing p and the units used to measure of program size determines the asymptotic slope of the constant efficiency curves (on a log log scale). If $C(p, K)$ grows with p then we will not be able to increase p and K indefinitely while keeping time constant. The slope of the constant efficiency curves determines how the problems will scale. Small values of the slope will mean that if we increase problem size linearly with processors we tend to increase efficiency and and large values mean that we will tend to decrease efficiency.

5 Summary

In order to be able to determine how problem size must increase with the number of processors we express performance and efficiency as a function of problem size and number of processors. We found that in regions low efficiency adding processors will not significantly increase performance. Therefore we must scale up the problem in such a way as to avoid this region. For the simple case when the communication cost is constant, this region is bounded by a straight line. For the general case the region of low efficiency is bounded by a curve, say $E = .5$ which in many cases may be asymptotic to line on when plotted on a log log scale. A "steep" constant efficiency curve (one which has slope > 1 on a log log scale) means that problem size must increase faster than the number of processors to maintain the same efficiency and that the algorithm will not scale well.

The hardware and software characteristics will determine the average time to perform an operation, and therefore peak performance $r_{\infty}^{(p)}$. Decreasing τ together with τ_S and τ_C will change move the constant performance curves down and to the left (compare Figure 3 and Figure 1) The parameter $k_{1/2}$ determine the rate at which we reach the asymptotic performance $r_{\infty}^{(p)}$ and $r_{\infty}^{(K)}$ (compare Figure 2 and Figure 1). $k_{1/2}$ is determined by the relative speed and number of sequential and communication operations. The rate of growth of $C(p, K)$ with p and K determine the slope of the constant efficiency curves which in turn determine how the problem will scale. $C(p, K)$ is a characteristic of the algorithm, and the architecture influences the choice of algorithm. There may be a more meaningful measure, N, of problem size than K and if the problem is expressed in terms of N, the scaling characteristics may change.

We have also indicated how these constant performance curves can be used to compare different architectures as disparate as the Cray and Connection machine by rescaling the p axis.

This model does not directly address the question of whether massive parallelism is feasible. It provides a framework for evaluating how particular algorithms will scale. If for many algorithms of interest $C(p, K)$ grows rapidly with p or K, massive parallelism may not be feasible unless the communication can be overlapped with computation. If $C(p, K)$ grows sufficiently slowly with p or K, or if the coefficients in $C(p, K)$ are be sufficiently small, the asymptotic behavior may not be evident when we scale the problem. We may consider the case where $C(p, K)$ is $O(p^{\alpha})$ to be a perturbation of of the case where $C(p, K)$ is constant, and if the perturbation is sufficiently small we may achieve our desired performance without running into the effects of Amdahl's law. This is especially true if $C(p, K)$ is $O(log(p)$. What is needed is experimental data for a large number of algorithms on present machines. We can estimate how the algorithms will scale by extrapolating the line (or curve) that goes through the "vertices" of the constant performance curves and these estimations may shed light on the whether massive parallelism is feasible.

Acknowledgment I wish to thank Eric Barszsz and Horst Simon for correcting several errors and suggesting improvements in substance and presentation.

References

[1] Amdahl, Gene M., *Validity of the single-processor approach to achieving large scale computing capabilities*, AFIPS conf. proc., 30(1967), 483-485.

[2] Amdahl, Gene M. , *Limits of Expectation*, International Journal of Supercomputing Applications, 2(2) (1988), 88-94.

[3] Flatt, Horace and Kennedy, Ken, *Performance of parallel processors*, Parallel Computing 12(1989), 1-20.

[4] Fox, G. C. and Otto, S.W., *Matrix algorithms on a hypercube I: Matrix Multiplication*, Parallel Computing 4 (1987) 17-31.

[5] Gustafson, John L, Montry, Gary R. and Benner, Robert E., *Development of Parallel Methods for a 1024-Processor Hypercube*, Siam J. Sci. Stat. Comput., 9(4) (July 1988), 609–638.

[6] Hack, James J. *On the promise of general-purpose parallel computing*, Parallel Computing, 10(1989), 261–275.

[7] Hockney, R. W. and Jessope, C. R., *Parallel Computers 2,* , Adam Hilger, 1988.

[8] Patton, Peter C., *Performance Limits for Parallel Processors*, in "Parallel Subercomputing: Methods, Algorithms, and Applications", Graham, Curey ed, John Wiley and Sons, 1989

[9] Van-Catledge, Frederic A., *Toward a General Model for Evaluating the Relative Performance of Computer Systems*, The International Journal of Supercomputer Applications, 3(2) (1989), 100–107.

[10] Worley, Patrick H., *Limits on Parallelism in the Numerical Solution of Linear PDES*, Oak Ridge National Labatory, Technical Report ONRL/TM-10945 (1988).